ELSEVIER

Computers and Electronics in Agriculture
28 (2000) 29–49

Computers
and electronics
in agriculture

www.elsevier.com/locate/compag

# Object-oriented simulation of integrated whole farms: GPFARM framework

M.J. Shaffer *, P.N.S. Bartling, J.C. Ascough II

*USDA-ARS, Great Plains Systems Research Unit, P.O. Box E, Fort Collins, CO 80522, USA*

Received 1 November 1999; received in revised form 13 January 2000; accepted 15 February 2000

**Abstract**

Simulation frameworks for Decision Support Systems (DSSs) at the whole-farm level have not adequately supported process level management and integration of complex farming systems. Development of the Great Plains Framework for Agricultural Resource Management (GPFARM) DSS for whole-farm management required a simulation package that could handle this complex system. Object-oriented (OO) techniques now offer improved opportunity for the development of suitable whole-farm simulations. A whole-farm simulation framework was developed using Object-oriented programming (OOP) and executes appropriate simulation modules, written in procedural languages (FORTRAN and BASIC), for the lower-level processes. Abstraction, encapsulation, and hierarchy were crucial to simplifying whole-farm complexity. OOP relationships, particularly inheritance in crops, animals, and events were key in allowing dynamic (runtime) setup and simulation of the farm system. Event objects were positioned outside of the simulation class and farm state to provide event input format flexibility and allow the framework to check the system state before events were implemented. The inclusion of existing or extended procedural modules cut development time, helped insure maintenance support from cooperators, and assisted with deployment of computationally efficient code. The resulting package simulates whole-farm management involving interactive land units, integrated crop and livestock operations, crop rotation systems and multiple commodities, and soil and climate variability. Simulation of whole-farm scenarios provides information for DSS applications to display and compare management alternatives. This research showed that an effective OO framework for integrated farming systems should consist of a thorough object model of a farm state, a flexible input and implementation of events, and a simulation environment to accomplish the biological, chemical, and physical simulation of a farm.

* Corresponding author. Fax: + 1-970-4908310.

*E-mail address:* shaffer@gpsr.colostate.edu (M.J. Shaffer). Published by Elsevier Science B.V.

0168-1699/00/$ - see front matter Published by Elsevier Science B.V.
PII: S0168-1699(00)00117-4

*Keywords:* Whole-farm management; Soil-crop models; Ranching; Environmental quality; Decision support systems

## 1. Introduction

A whole-farm Decision Support System (DSS) should provide the agricultural manager with information about resources across the farm and potential impacts of management decisions on those resources. To accomplish this, a DSS like the Great Plains Framework for Agricultural Resource Management (GPFARM) must include many components to supply information to the user. The GPFARM DSS components assisting users include: a Graphical User Interface (GUI) for input, a scenario manager, economic and environmental simulation models, site database generators, an information system, output visualization, and indices to compare results (Shaffer and Brodahl, 1998; Ascough et al., 2000). In particular, GPFARM must rely on a rigorous integration framework and process-based simulation package that can handle the complexity of the system. Many complex issues must be addressed before a viable whole-farm DSS can be designed or constructed. The presence of multiple land units and uses; interactions across the farm/ranch; production of multiple commodities; and variable soils, climate and management activities are a few examples.

Traditionally, simulation of the soil–plant–atmosphere system involved reading input for a site, implementing management events through time, and calling process level science models. These models yielded single point data and were evaluated for performance at a location (Khakural and Robert, 1993; Shaffer et al., 1994; Beckie et al., 1995). More recently, models have been used in conjunction with geographic information systems (GIS) and executed multiple times for points across a landscape to give spatial simulation results (Shaffer et al., 1995; Engel et al., 1997). However, a dynamic spatial link generally has been absent, that is, having the simulation at one location impact other locations at each time step. This link is required for whole farm simulation because farm parts often share or transfer resources affecting their simulation.

Whole farms consist of many components that may be sharing or competing for resources. This complex array of components and their interactions is challenging to program, especially using procedural languages such as FORTRAN and BASIC. As a result, there is a lack of effective whole-farm simulation models. Rather, the tendency has been to create convoluted code that is difficult to interpret and debug, and cannot be easily extended or ported to related applications. Object-oriented programming (OOP) methodology was developed with the intent of alleviating some of these problems, but not without the introduction of a coincident, steep learning curve for FORTRAN and BASIC programmers. However, the specific benefits of OOP methods: data organization, code reuse, and code flexibility appear promising especially with extremely complex systems such as whole-farms.

OOP methodology applied to simulation in agriculture is not new. For example, OOP applications have been developed for specific parts of the system such as field operations, nitrogen dynamics, and individual plant growth (Crosby, 1990; Sequeira, 1990; Lal, 1991; Freeman, 1992). Object-oriented (OO) frameworks for information delivery in DSSs have been implemented in conjunction with OO databases and expert systems (Power, 1993; Van Evert and Campbell, 1994; Gauthier and Neel, 1996). Whole farm modeling with the use of heuristic based constraint satisfaction or linear programming approaches have also been documented previously (Buick, et al., 1992; Pannel, 1996; Schilizzi and Boulier, 1997).

A fully object-oriented simulation of a large agricultural system like a whole production farm with multiple products and environmental impacts is lacking. Silvert (1993) suggests the reason is that most process-level models are written in procedural languages, and the code would need to be redesigned and rewritten before it could be used directly in object-oriented modeling. This was a key issue in the development of the GPFARM environmental simulation model. Contributing scientists had considerable time invested in their respective science process modules written in FORTRAN and BASIC. Therefore, an Object-oriented (OO) simulation framework was developed that describes key farm system components and interactions, and provides a medium for simulation. OO system analysis, design, and programming were key in simplifying the complex farm, and increasing simulation flexibility while making use of existing FORTRAN and BASIC code for lower-level process simulation.

The OO approach in simulation allowed GPFARM DSS to meet its objectives of providing the model user (agricultural producers and consultants) with information for a whole-farm analysis based on interactions between cropping systems, animals, weather, soils, and other management components in terms of bottom line economics and environmental impacts. The objective of this paper is to present the development of an OO framework for simulation at the whole-farm/ranch level with emphasis on the GPFARM Simulation Framework. The primary challenge was the development of an OO structure for a whole-farm suitable for inclusion in a DSS, and yet also compatible with process-level simulation.

## 2. The object-oriented farm simulation

An object-oriented analysis, design, and programming approach was used in the GPFARM Simulation Framework. Object-oriented analysis (OOA) examines ‘What’ will be in the model and uses classes to describe objects defining the farm. Object-oriented design (OOD) addresses ‘How’ the model will work focusing on relationships and interactions on the farm. Object-oriented programming (OOP) is the method of implementation (Booch, 1994). There are many ways to accomplish OOA and OOD. The Object Modeling Technique (OMT), the Object Oriented Software Engineering (OOSE) method, and the Booch Method are typical examples (Rumbaugh et al., 1991; Jacobson et al., 1992; Booch, 1994). The Booch Method was used for the GPFARM Simulation Framework; and simplified Booch Lite notation is used for the object-oriented design figures in this paper (Booch, 1994). The programming was implemented using C++. Four of seven elements explained in the object model must be present to be OOP; they are abstraction, encapsulation, modularity and hierarchy (Booch, 1994). These are accomplished by organizing data and characterizing them by classes, hiding data, partitioning the program with well defined boundaries, and establishing inheritance relationships, respectively. Relationships among objects are a key concept in OOP. The framework makes extensive use of the 'Is a' (generalization/specialization), 'Has a' (whole/part) and 'Uses' (association) relationships (Booch, 1994). The simulation framework and DSS simulation objectives resonate throughout this approach. They are: to be computationally efficient for fast execution, minimize input or hide complexity from the user, maintain enough detail to simulate effects of changing management on resources, and allow flexibility in the farming systems simulated.

A good abstraction can simplify the system complexity and emphasize details that are significant to the user and suppress those details, for the moment, that are diversionary (Shaw, 1984). As applied to an integrated farm (a place having activities dealing with the cultivation of land and raising of animals) simulation, abstractions must bring out key concepts important to the farmer and scientist. For the farmer, the focus is on products and resources, namely the crop produced, the herd of animals, or the land condition. The scientist focuses on accurate simulation of those objects and adds others like climate, crop parameters, breed traits, and pesticide parameters to name a few. The number of potential objects may be as complex as the original problem. Specifically, abstraction, encapsulation, and hierarchy are crucial to simplifying whole farm complexity. Modularity and the dynamic (at run time) creation of objects greatly enhance the simulation's flexibility to handle a changing farm structure as well as changes in programming and simulation technology.

The simulation framework objects can be classified into three categories for discussion purposes, Fig. 1. The details of the categories and associated classes characterizing framework simulation objects are presented in Table 1. The farm state category includes objects describing the farm system that can be physical (quasi-static places, meaning static at least in location) or dynamic (elements always changing in type or location) in nature. The interactions category includes objects describing events or interactions across the farm, and the simulation category represents objects executing process level science modules. The GPFARM simulation framework uses these objects to maintain the status of the farm system, create and implement events, and iterate time and space while creating simulation environments to execute and integrate FORTRAN and BASIC process level science modules. The science modules are: nutrient cycling and residues (Shaffer et al., 1991; Hansen et al., 1995), weed impacts on crop yield (Wiles et al., 1996; Canner et al., 1998), water/solute transport and soil properties (Nachabe and Ahuja, 1996; Ahuja et al., 1999), forage and animal production (Baker et al., 1992; Hanson et al., 1992) crop growth (Williams et al., 1984; Arnold et al., 1995), evapotranspiration (Farahani and Ahuja, 1996; Farahani and DeCoursey, 1999), and soil loss due to erosion (Ascough et al., 1995, 1997).

### 2.1. The farm state category

In general, a farm state category includes all objects that hold the status of the farm for the simulation. Objects are, by one definition, structures, locations, roles played or events (Coad and Yourdon, 1991). The farm state has both physical places (objects static in location) and dynamic players (objects not static in type or location). The physical objects include the farm, subfarm, field, management unit, and the various layers making up the environmental layer system. The dynamic elements or players, which change in type or move across the farm, include crops and animals (Fig. 1).

### 2.2. Farm state physical character

#### 2.2.1. Farm

The farm place has many features associated with it. Some physical farm features include fields and pastures, while others are man-made like storage silos and corrals. Some require significant simulation be preformed on them. Others are present to account for storage and use of resources on the farm. The role of a farm in this framework is to hold information about all of its parts so they can be simulated. Fig. 2 shows the Farm's parts through the 'Has a' relationships.

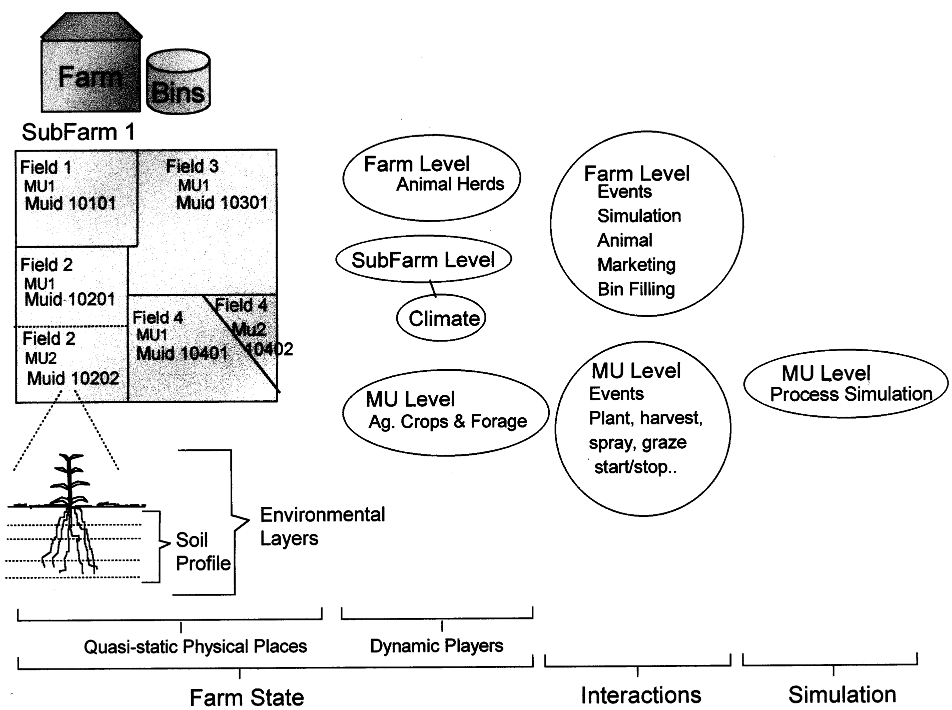


Fig. 1. Components of an integrated whole farm.

Table 1
Classes of a whole farm simulation framework

| Category | Nature | Class name for object abstraction | Definition (maximum number of objects as applied in GPFARM 1.0) |
|---|---|---|---|
| Farm state |  | *Farm* | The entire farm place to be simulated (1) |
| (Physical place) | Quasi-static | Bins | Storage places for feed and manure (6) |
|  |  | Subfarm | Corresponds to different climates (3) |
|  |  | Climate | Climate station data object (3) |
|  |  | Climdat | Daily climate data objects (days * 50 years) |
|  |  | *Mu* | Management Unit Level smallest area simulated (60) |
|  |  | Subfarmid | Subfarm location identification |
|  |  | Fieldid | Field location identification |
|  |  | Canopylayer | Respiring layer of the MU (1) |
|  |  | Surfacelayer | The surface of the soil (1) |
|  |  | Pestiapp | Pesticide apps. Residing the soil surface (40) |
|  |  | SurResidapp | Residue group residing on the soils surface, i.e. crops, manures, or other organics (3) |
|  |  | Residapp | Residue applications (80) |
|  |  | Soil | The soil profile (1) |
|  |  | Soillayer | Soil layers consisting of depth to 7.5 cm, 15 cm, $\leq$30cm, rooting and profile depths (5) |
|  |  | Pesticide | Pesticide apps. From surface left in soil layer (40) |
|  |  | Lresidapp | Part of residue from surface left in soil layer (3) |
|  |  | Residapp | Portion of residue apps. in soil layer (80) |
| (Players) | Dynamic | *Farm* |  |
|  |  | DomHerd | A domestic herd of animals, i.e. cow–calf herd (1) |
|  |  | Animals | Contains average statistics for animals (5) |
|  |  | Traits | The animal breed traits (1) |
|  |  | Supavail | The times when supplement is available to animal (5) |
|  |  | *Mu* |  |
|  |  | Agricrop | A cultivated crop for grain or hay (1) |
|  |  | Crpparam | Parameters for crop growth (1) |
|  |  | Foragecrop | A native rangeland (1) |
|  |  | Forageclass | Forage components including: cool and warm season grasses, forbs, shrubs, and legumes (5) |
| Interactions | Dynamic | *Event* | Events can be management or interactions(unlimited) |
| (Events) | (Farm level) | Pendingevt | Generic pending event |
|  |  | Feedsupevt | Initializes farm feed supplement bins |
|  |  | Precipevt | Precipitation event adds nutrients found in rainfall |

Table 1 (*Continued*)

| Category | Nature | Class name for object abstraction | Definition (maximum number of objects as applied in GPFARM 1.0) |
|---|---|---|---|
| | (Mu level) | Plantevt | Planting event for cultivated crops |
| | | Harvestevt | Harvest event for cultivated crops |
| | | Tillageevt | Tillage event for cropland unit |
| | | Herbevt | Herbicide application event for cropland unit |
| | | Irrigevt | Irrigation event for cultivated crops |
| | | Grazestrtevt | Herd movement event on rangeland management unit |
| | | Grazeendevt | Herd movement event off rangeland management unit |
| | | Nutricevt | A chemical nutrient application |
| | | Nutrimevt | A manure nutrient application |
| Simulation | Simulation unit | *Su* | Framework simulation class which calls FORTRAN libraries and initiates BASIC applications |

### *2.2.2. Subfarm*

The farm can physically be widely distributed, having parts of the farm in different climate regions. For example, a farmer may want to use the GPFARM DSS to look at whole farm economics, environmental impacts, and production across mountain grazing areas as well as crop production on the plains. The subfarm concept was designed to address simulating areas with different climates. A subfarm identification is attached to the land unit to load the appropriate daily climate for simulation of that unit. In GPFARM 1.0, the framework is designed to handle up to three subfarms. Each subfarm reference identifies a position in a dynamic template array of climates. Each climate template holds climate station data as daily climate for the simulation.

### *2.2.3. Field*

Farmers generally define their land units as fields or pastures. Roads, fences or natural features like streams usually delineate these land areas, and management within can be constant or variable. The field is a key concept for the manager to locate land units on the farm and identify their land use. However, this is an extremely variable environment where soil conditions, weed pressures, and landscape characteristics impact simulation outcomes significantly even with a given land use. Smaller delineated land units (management units) are necessary to hold information specific enough to simulate variability across the field. However, maintaining the field as a location tag in the farm state category of the framework is crucial to the presentation and aggregation of simulation results for the user. Therefore, management unit (Mu) ‘Has’ a FIELDID (Fig. 2).

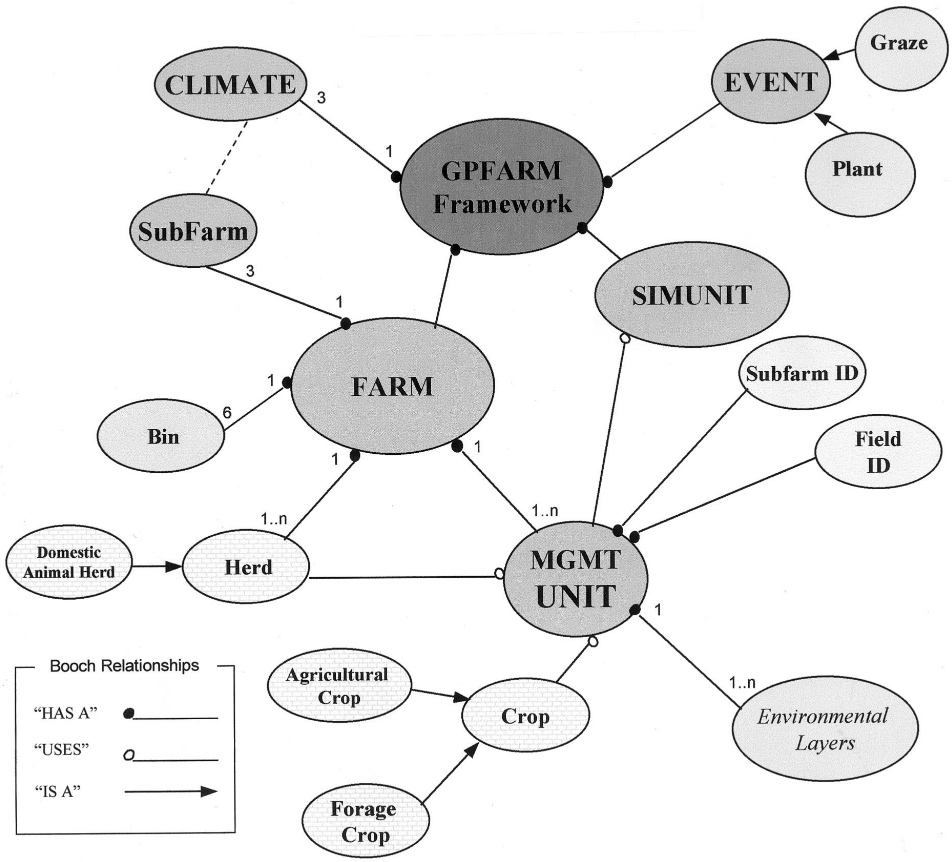


Fig. 2. Object-oriented design of GPFARM 1.0 simulation framework.

### 2.2.4. *Management unit*

The management unit (Mu) is a key physical land area to delineate for simulation. Mu delineation for the purposes of simulation varies with the user's application and needs. For example, for a farmer in precision agriculture or a soil scientist, a Mu may be a land area where one management scheme dominates on a given soil. For a ranching application, the management scheme, rather than soil normally drive a Mu delineation. Implementation of a grazing rotation is dependent on physical barriers to manage the location of animals. The hydrologist may delineate a Mu by the shape of the landscape where slope defines flows over a larger area. The Mu in the GPFARM simulation framework is designed to hold data ranging from the general to the more specific land delineations.

The Mu's role in the farm state is to contain all the biological, physical, and chemical data necessary for the environment to be simulated. It provides functions to process user input data for simulation. The Mu class makes objects of its parts, establishes relationships among other objects, loads initial condition information into its parts, and contains special functions for the soil system. These functions include depth weighting incoming soil layer data to a simulation soil-layer system, and calculating the status of the soil water.

The Mu is the smallest unit to be simulated and results can be output to the user or transferred during runtime (dynamically) to another Mu as input. For example, interactions among Mu's in water runoff and runon can be handled with interface input on Mu flow dependencies and an event implementation defined for when runoff occurs on a Mu. The event implementation defined by user or the framework would dictate the timing of runoff as run-on to another unit.

The framework simulates each Mu daily and writes output from it. The framework relies on other applications for output aggregation. In GPFARM 1.0, separate post-processing applications such as the GPFARM Visualization or GPFARM economics package aggregate and process Mu biological, chemical and physical output in a manner and at the level the user wishes to see.

*2.2.5. Mu and environmental layer system*

Fig. 1 shows a schematic representation of an integrated farm where parts depicted assist in abstracting the Mu class. OO Design phases demonstrated that a Mu has area extent, location, environmental layers, and 'use' associations with crops and animals (Fig. 2). Further abstraction of the environmental layers for GPFARM 1.0, resulted in a Mu having one canopy layer, one surface layer, one soil profile, and up to five soil layers (Fig. 3). Table 1 shows the Mu class structure (indentations representing 'has parts' relationships with other class objects).

The Canopylayer object holds state variable information with regards to canopy dynamics. These include wind speed in the canopy, light interception, canopy humidity, and arial and basal covers. The object holds potential and actual evaporation and transpiration for the grain or forage crop. Many of the plant attributes residing in this layer such as height, leaf area index, and cover and are assigned to a crop base class discussed in the dynamic (changing in type or location) aspect farm-state objects.

The Surfacelayer object contains all data characterizing the soil surface, such as residues and pesticides present, topography, roughness, reflectance, infiltration attributes, erosivity, and functions to aggregate some of its parts. These components are crucial to the soil nutrient budget and are affected by management decisions regarding tillage. In addition, the presence of residues and pesticides are tracked for each application through time. This seemingly complex accounting system is simplified into a few 'Has a' relationships with classes which can be utilized in the system above and below ground. For example, the Surfacelayer 'has' three types of SurResidapp which 'has' many Residapp objects (Table 1).

The soil profile object (ASoil) contains the condition of the soil profile. Input information for the soil series resource is stored here, such as layer restrictions, series name and texture, wet and dry albedo, and depth to water table and hard pan. A soil profile schematic has soil layers as its parts (Fig. 1). However, a model developer must judge whether further nesting of parts will add functionality or encumber it. For the GPFARM 1.0 simulation framework application where ten

FORTRAN/BASIC coded science modules would be loading data to and from the physical Mu object, another parts hierarchy provided more encumbrances than functionality; therefore, the Mu object has a soil and soillayer objects directly instead of nested (Fig. 3). In addition, the Mu class has a Nbudget object that is used to maintain variables calculating nutrient and water mass balance in the profile. It also holds additions of various forms of nutrients to the Mu from the implemented daily events for simulation.

Soil layers represented by the Soillayer class have variables describing both physical and chemical layer attributes. These include bulk density, pH, soil organic matter, nitrate nitrogen, ammonium nitrogen, water content, and root mass to name a few. As with a Surfacelayer object, each soil layer would contain any residue or pesticide objects that infiltrated into the soil from the surface applications through time (Table 1). Soil profile attributes likely to impact a manager's plan, such as root zone water and residual nitrates, are derived from these layer objects.

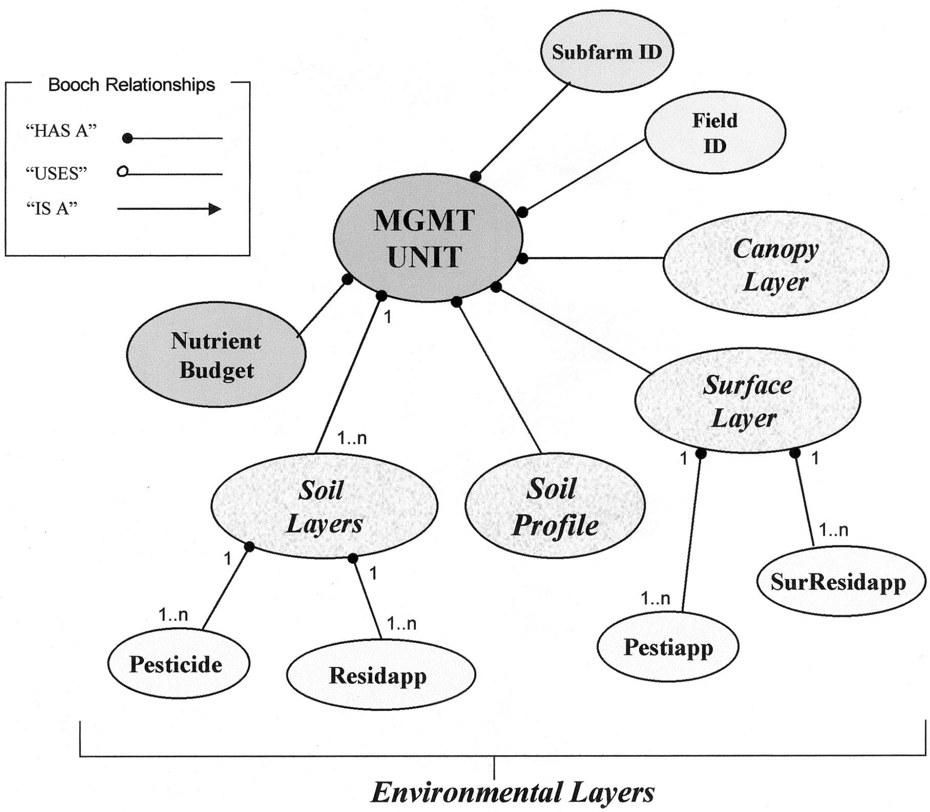


Fig. 3. Object-oriented design of Mu's environmental layers (canopylayer, surfacelayer, soil profile and soil layers).

### *2.3. Farm state dynamic character*

Dynamic character refers to the environmental players whose presence or nature changes with time on the farm. While the farm physical character holds data about places static in location, the farm dynamic character holds data about the changing and moving players on the farm.

#### *2.3.1. Environmental players*

Animal types and crop types are key examples of environmental players. Animals move across the farm and crop types are transitory, changing with each year in a crop rotation. These players are present and interact at different levels of the physical farm (Fig. 1). Farm physical places have associations or 'use' relationships with these players (Fig. 2). For example, a Mu object can use information stored with the farm level animal herd. Mu herds can be all or part of the farm herd and the herds can be domesticated or wild. The farm state base structure must be flexible enough to handle new players in the system, as new science modules become available to simulate them.

The animal kingdom is a complex aspect of the farm players needing simplification. There are both wild and domesticated animals that are either confined or free ranging; all impact the environment, use resources, and may produce revenues. A generalized base class, Herd has animal type and herd size as its variables and provides a good foundation class on which to derive herds for simulation. A specialized domestic herd (DomHerd) is a Herd (Fig. 2). In GPFARM 1.0, a cow–calf herd is the domestic herd on the farm but other herds could be derived if the application warranted it. The domestic herd has many parts including breed traits, animals, and several kinds of supplement available to it (Table 1). The farm object's role is to read the user provided information and if an animal herd is present, a Domherd is allocated in memory at runtime and passed arguments on herd type and animal size for the base class data requirements.

Farm players can also be abstracted from land use. The GPFARM crop players, Agricrop and Foragecrop, are created in the management unit object construction by reading a management unit land use which is either cropland or rangeland (Table 1). Classifying similarities into generalized objects and differences into specialized objects ('is a' relationship) accomplishes the framework's objectives in simplifying complexity with animals and plants, but also increases their simulation flexibility (Fig. 2). Framework simulation flexibility increases by allowing dynamic (runtime) execution of process simulation models depending on the specialized object present. For example, a base class crop provides the crop variables needed by the potential evapotranspiration and the water balance and chemical transport models. These variables include plant height, totals for live and dead leaf area index, canopy cover, and water and nitrogen stress. These same models run regardless of the specialization of the crop. However, when a specialized player is present, the appropriate plant growth model is called in the simulation environment (Simunit). If a Foragecrop object is present, the rangeland forage model will be called; and, if an Agricrop object is present, the crop growth model will be called.

In the same manner, if a Domherd object exists the animal growth, intake, and population models will be called.

### 2.4. Interactions — the event system

The simulation framework creates and implements event objects for management operations and interactions across the farm. The framework has access to the event and the condition of the farm and its parts before the event is implemented. It calls the event implementation function or delays it until the state is favorable. This concept gives users the flexibility of fixed date or rule-based management operations as a means to input events. The GPFARM 1.0 simulation framework handles farm events such as the marketing of animals and the maintenance of feed bins. It also creates and implements management operation objects for the Mu such as, planting, harvesting, tillage, weed control, and nutrient application events for cropland units. For rangeland management units, it allows herd on and off events. These events are all derived and inherit information from the base class Event using the 'is a', inheritance relationship (Fig. 2). In general, the Event object has a place (usually a Mu identification or Muid) and type of event (usually a four letter code). The date is conspicuously absent from the event base class. This is because rule-based events may be derived from this class and have no need of a date. In addition, fixed date events are read every day for implementation so that date storage is not necessary. The main concern of this approach was the time requirements for the interface to write an event file for 20 years and the framework to read it daily. However, three important elements in the simulation framework's development picture eased that concern. First, advancements in database system query language (SQL) and database class development like Open Data Base Connectivity (ODBC) or Data Access Object (DAO) have made it almost effortless to read a table, maintain position, and manipulate records. Secondly, reserving program memory to handle more simulation objects for the whole farm rather than storing event history is certainly desirable. Finally, once a rule-based management input system is attached and operational, writing input events for 20 years will soon be obsolete.

The framework can also link a rule-based management system developed concurrently with GPFARM (Shaffer and Brodahl, 1998). A Beta version of GPFARM ran with this rule-based management system applied as the only means of generating management operations for the simulation. Work is currently underway to fully implement this system to run concurrent with a fixed date system allowing the next GPFARM simulation framework application more mixed event input capabilities. The user could fix dates for certain events, but allow irrigation by a rule rather than the current system of fixed date or at a scheduled interval. An application interface object allows the rule script, generated from user interface input, runtime access to appropriate GPFARM state variables and transfers the action to be taken back to the simulation program as an event. For example, an irrigation application includes querying rules for the Mu and checking the water depletion status. If water depletion is below a user defined level, the application's interface object will be give an appropriate irrigation event (AnIrrigevt) back to the framework for the Mu and day. The same event format can be used by the fixed date system only it is initiated from an event table record written by the GPFARM DSS graphical user interface (GUI).

### *2.5. Simulation — the simulation environment*

The simulation environment or Simunit class is a key abstraction for the scientist and programmer to execute process level models. Process level models execute using daily climate, date and the Mu place, however not all models execute on every Mu or every day. For GPFARM 1.0 the overall suite of models include: potential evapotranspiration, water balance and chemical transport, soil properties, nutrients and residues, agricultural crop growth, forage crop growth, soil erosion, weed competition, herd dynamics, and animal growth and intake.

These models could have been part of the farm layer system with the canopy layer able to call the potential evapotranspiration model. However, many scientists were contributing these models to GPFARM in different languages and programming styles. Isolating them to their own object seemed prudent. Often when existing programs are put within other applications they are put in a wrapper. Isolating submodels in their own object increases programming flexibility to replace components, as better options become available.

### *2.6. Implementation of the integrated farm simulation framework*

The simulation framework is a C++ executable called from a dialog box in the GPFARM Graphical User Interface (GUI). The simulation framework makes the farm place and dynamically allocates the climate template array. For each day and place, it reads events or queries a rule system (resulting in events), and implements events in the Mu place. All interactions between Mus are considered as events and are implemented daily before the Mu is sent to simulation. The framework performs the simulation by making the simulation environment, ASu (Fig. 4, part D). It also reports the status of the farm at crucial management times throughout the simulation.

#### *2.6.1. Creation of the farm system*

The creation of the GPFARM whole farm place is accomplished in one statement, ‘Farm AFarm’; (Fig. 4, Part A). The physical place and the dynamic players of the farm are built by creating objects, data and function hiding, expressing relationships including ‘has a’ and ‘uses’ and establishing a hierarchy through inheritance with the ‘is a’ relationship. All essential aspects of object-oriented programming which when implemented in the agricultural system state takes the complex and makes it simple.

```
PART A
//CREATE A FARM STATE , LOAD FARM DATA, SUBFARM INFO, CREATE AND LOAD MU'S
FARM AFARM(TOT_SF); . . .

//CREATE PROGRAM DATE OBJECTS FOR STARTING DATE, SIMULATION AND ENDING DATE
PROGRAM_DATE STARTDATE; . . .
//CREATE DYNAMIC TEMPLATE ARRAY OF CLIMATES NEEDED AND LET IT LOAD DATA
CDYNARRAY <CLIMATE, 3> RAY;
 FOR (INT NINDEX = 0; NINDEX < 1; NINDEX++){CLIMATE T (. . .) ; RAY.ADDTOEND(&T); }

//EVENT DECLARATIONS
PENDINGEVT  APENDINGEVT;
PLANTEVT APLANTEVT; . . .

//FIXED EVENTS: READ FIRST PENDING EVENT DATE, TYPE AND DATA
M_PEVTSET->MOVENEXT();
LOADEVENT(M_PEVTSET,APENDINGEVT); . . .

PART B
//SIMULATION DAILY LOOP
WHILE  (SIMDATE <= STOPDATE)
{. . .

//SIMULATION EACH MU LOOP WITH IN EACH DAY
FOR (MUCNT=0;MUCNT < AFARM.TOT_MU;MUCNT++)
            {. . .
            WHILE (APENDINGEVT.EDATE() == SIMDATE && APENDINGEVT.GETMUID()==
              AFARM.MU[MUCNT].MUID())
                        {        . . .
PART C                              IF(  APENDINGEVT.GETEVTTYPE()=="PLNT")
                                    {
                                    APLANTEVT=APENDINGEVT;
                                    //REPORT SYSTEM STATUS AT PLANTING TIME
                                    WRITEPLANTOUT(&AFARM,&SIMDATE, MUCNT,
                                    APLANTEVT. . .);
                                    //CALL EVENT IMPLEMENTATION
                                    APLANTEVT.IMPLEMENT(AFARM.MU[MUCNT]);
                                    //MOVE NEXT RECORD, LOAD INTO PENDING EVENT,
                                                IF(M_PEVTSET->ISEOF()==0)
                                                {
                                                M_PEVTSET->MOVENEXT();
                                                LOADEVENT(M_PEVTSET...);
                                                }
                                    }//END PLANTING
                                                . . .
                        } //END WHILE EVENT DATE and PLACE == CURRENT

PART D
//CREATE A SIMUNIT Object
SU ASU (&AFARM,&AFARM.MU[MUCNT],RAY.GETAT(SFID), MUCNT . . . ) ;
}//END MU LOOP

//OTHER REPORTING
        IF (SIMDATE.LASTDYOFYR()) {  WRITEANNUALOUT(&AFARM,&SIMDATE);}
++SIMDAY; ++SIMDATE;
}  //END DAILY SIMULATION WHILE LOOP
```

Fig. 4. C++ code implementation of the GPFARM simulation framework.

### 2.6.2. Time iteration

Program_Date objects are made for the starting, ending, and simulation dates of the run (Fig. 4, Part A). These objects inherit standard Date accounting from the base class Date but also provide functions for testing end-of-month and end-of-year useful in output reporting. It is incremented daily in a While loop as long as Simdate is less than or equal to Stopdate (Fig. 4, Part B).

### 2.6.3. Space iteration

Daily iteration of space is implemented in a For loop for all Mu's on the farm (Fig. 4, Part B). This aspect was debated as to whether the load and save on each Mu each day would slow the simulation too much. The overwhelming advantage was the possible interaction among Mus on a daily basis. This meant transfer of water, nutrients, residues and other resources or products from one Mu to the next over a field or a farm was possible during runtime (dynamically). For example, Mu runoff and erosion output could be routed, given some Mu routing scheme from the user, thus creating a watershed simulation environment. With the current number (60) of allowable Mus in the program, such iteration affords more land unit interaction advantages than limitations.

### 2.6.4. Implementation of the events

For each Mu all fixed date events are read and implemented. A fixed date event is read from presorted Microsoft Access tables first into a pending event where type of event is read. Based on the type of event, the event object is filled and implemented before the Mu is passed to the simulation unit (Fig. 4, Part C).

### 2.6.5. Creation of the simulation unit

The simulation of the Mu occurs when a simulation object (ASu) is made (Fig. 4, Part D). The Simunit object is passed farm resources like cattle herds and supplement bins, a Mu, and a daily climate in its construction. Having all required agricultural system state information to process, the FORTRAN variable connections are established, and nine FORTRAN models and one BASIC science model are called depending on the dynamic players present. All newly simulated information is saved back to the agricultural system place.

### 2.6.6. Reporting on the farm

Reporting the status of the farm can be initiated when the simulation reaches predefined crucial dates; for example, the last day of the year for annual output (Fig. 4, Part D). Often it is initiated by an event (i.e. plant, harvest) and can be included with the event implementation or split into its own function and called after the event implementation. As an example, the soil profile status of each Mu is written on the planting date (Fig. 4, Part C).

## 2.7. GPFARM connection between interface and simulation framework

The connection between the GPFARM GUI and the science framework utilizes Microsoft Access databases to pass information. Fig. 5 illustrates the connection. The user enters the required input information (e.g. equipment, investments, Mu landuse and area, Mu resources and management operations, and climate) in the GUI. The simulation framework may then be run by selecting a 'Run Simulation' dialog box in the GUI (Fig. 5, Block 1). At this time, the scenario(s) to be run and the time range for the simulation are also specified. A scenario is a farm setup with a given management plan and resources. A farmer can set up multiple scenarios with the DSS and simulate management plans before implementing them.

All input tables, necessary to run the GPFARM science model, are contained in the scenario Microsoft Access database USERDB. The GUI fills the USERDB tables as the next step (Fig. 5, Block 2). The major tables filled contain information on farm name and location; farm equipment and investments; Mu area and landuse; Mu soils, topography, residue, and initial weed population; and Mu cropping systems and management operations. Other tables are filled depending on additional options chosen within the GUI. For example, if weed management is chosen, tables containing information on weed pressure, pesticide efficacies, and pesticide properties (e.g. half-life) are filled. If rangeland is chosen for any of the Mu's, then tables containing information on herd description, forage and supplemental feed properties, and herd physical characteristics are filled. If the Mu is to be irrigated, then tables containing information on the irrigation system(s) are also populated.

The GPFARM science framework simulation is then executed (Fig. 5, Block 3) after USERDB has been filled. The CLIMDB (climate) input tables (Fig. 5, Block 6) and the USERDB input tables are used as input to the GPFARM simulation framework application. The OUTPUTDB (output) tables (Fig. 5, Block 7) are filled as the simulation is run. The major tables filled contain information on water runoff, nitrate leaching, wind and water erosion, and crop yield. If weed management is chosen, then tables containing information on yield loss due to weed pressure, and pesticide runoff and leaching are filled. If rangeland is chosen for any of the Mu's, then tables containing information on animal average daily gain, sales, forage and supplement consumption and peak forage biomass are filled.

After the science simulation framework is run, the economic simulation is performed (Fig. 5, Block 4). The economic simulation consists of two phases. First, required economic information is obtained from the equipment and investments database tables. Operating cost calculations are then conducted for equipment and investments on a per-hour basis. Second, the management operations for each Mu are examined one-by-one and per-hour operating cost calculations applied wherever applicable. In cases where there are also material costs (e.g. seed cost), the material

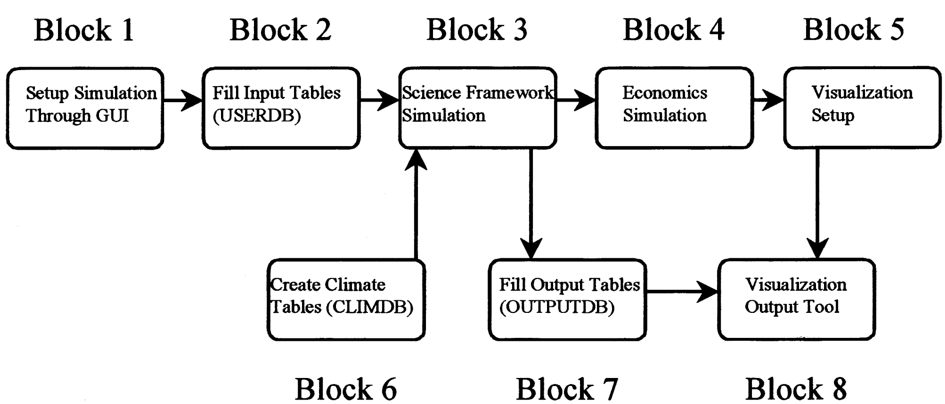


Fig. 5. Overview of GPFARM connection between GUI and simulation framework.

costs are directly combined with the operating costs. Crop yield and market price information are then used to calculate returns. Once all Mu economic calculations are performed, information on ownership costs, operating costs, returns on operating costs, and returns over total costs are placed into economic output database tables. These tables, along with some USERDB input tables, are used for the creation of detailed economic reports.

The visualization setup (Fig. 5, Block 5) uses the information gathered in the USERDB and OUTPUTDB tables to create a set of ASCII files that are used for quick access by the visualization output tool. The user may then select variables to be displayed in the visualization output tool at the level of farm aggregation they wish to see (Fig. 5, Block 8). Results from multiple management scenarios (simulations) may be compared with this tool.

## 3. Discussion and conclusions

In general, an effective OO framework for integrated farming systems should consist of a thorough object model of a farm state, a flexible input and implementation of events, a simulation environment to accomplish the process level simulation of the biological, chemical, and physical components of a farm, and iteration of time and space to allow interactions among spatial units and resources on the farm. The benefits realized from an OOP approach in whole farm simulation include simplifying system complexity and increasing flexibility of the simulation application. The object model together with OOP relationships, that is-whole/part, generalization/ specialization (inheritance), works to achieve these benefits. In addition, the OOP approach makes addition and reconfiguration of Mus and their interactions a much simpler and compact process than would be needed using other programming approaches. This allows potential users, such as farmers and consultants, to utilize the model as a close approximation of their farming or ranching operations. They may then draw conclusions from the simulation results and from economic and environmental analyses to help refine or define operations that maximize profits, yet protect the environment.

The magnitude of benefits from OOP however is dependent on its judicious use (Power, 1993). Whether an organism traverses the farm as an animal herd or resides at a location like a plant, the organism itself is as complex as the environment in which it exists. Simplifying this complex system starts with finding similarities, differences and subparts and bringing them out in the design. One of the most difficult design judgements to make is whether the organisms' attributes are different enough to abstract them to a new derived class. Certainly, process level models and their capabilities influence design abstractions. For example, one simulation model for forage production and one for annual crops calls for at least two objects. However, developers should avoid letting simulation capabilities drive the framework design, as new models will always present themselves requiring changes in the design. As an example, the GPFARM simulation framework definition of a domestic herd class is biased toward animal operations producing meat and not broad enough to cover domestic animals kept for other products. A good framework modification to simulate goats would be to make the domestic herd class (DomHerd) more generic and derive GoatDomHerd and Cattle-DomHerd from it.

Flexibility in simulation indicates a framework is responsive to change. These changes can be in the areas of farm structure and resources, input/output formats, farm management events, or simulation modeling techniques. The OO framework directly addresses changes in farm structure and resources through the dynamic (runtime) allocation of objects on the farm. The positioning of event objects outside the simulation class or farm object provides input/output format flexibility for events as well as allowing the framework to check farm state before events are implemented. Therefore, the framework can implement events in a practical manner similar to the methods of a farm manager in the field. The capability of a framework to handle fixed date and rule-based events will bring management event simulation closer to emulating the farmers own decisions. How close could only be answered by the attached rule system and its capabilities to handle complex rules entered by the farmer.

Previously, the importance of using inheritance and its advantages in reuse of code was explored (Silvert, 1993; Power, 1993). These previous findings proved valid in the GPFARM 1.0 simulation framework flexibility. The inheritance relationships in crops, animals and events were key in allowing dynamic (runtime) setup and simulation of the farm system. The subdivision of crops into agricultural and forage was particularly useful in executing their different simulation models.

Unfortunately, first-time designs are often skewed by the current simulation capabilities of the submodels. For example, GPFARM 1.0 science modules could not handle herds grazing actively growing crops like winter wheat. Fortunately, OOP approaches provide tools to handle this; such as, inheritance (to extend the current Agricrop to a grazed derived class) or polymorphism (to define a graze function to mean a separate implementation for each derived crop class). The framework will likely be expanded using these OOP tools as the need arises.

It has been also suggested that 'frameworks provide the easiest and most productive bases for effective reuse of existing software' (Gauthier and Neel, 1996). This framework did provide a good base for the effective use of pre-existing process level models; and, the decision to allow incorporation of procedural based FORTRAN and BASIC process simulation modules has proven beneficial in several ways. The scientists who developed these modules continue to support and extend their code used in GPFARM. The use of the minimal wrapper concept to execute these modules in the C++ object-oriented environment has allowed a more rapid turn around time when maintenance and updating are needed. However, the existing wrappers could be enhanced to make module additions and replacements even more streamlined. The addition of new simulation capabilities is made easier by the availability of a large array of FORTRAN and BASIC soil-plant process modules compared to object-oriented C++ equivalents. Although the contents of each FORTRAN or BASIC module are procedural, each module can be treated as an individual object or related modules can be grouped and treated as an object by the framework. Judicious grouping of process simulation modules can improve run times for the overall simulation. Also, in many cases, the FORTRAN versions of algorithms used to solve sets of equations are more efficient than their C++ equivalents.

## 4. Availability

The GPFARM simulation framework application 1.0 is currently operational within the GPFARM decision support system 1.0 in limited release to collaborating Colorado farmers and agricultural consultants. Although evaluation and validation tests are on-going, the integrated whole-farm simulation, the DSS, and its features have already proven useful in assisting managers with strategic farm planning. Additional information is available on the USDA-ARS, Great Plains Systems Research Unit, Internet Web site at URL: http://www.gpsr.colostate.edu.

## Acknowledgements

Many thanks to Mary Brodahl for collaborating discussions on object-oriented event design, rule based event systems, and her authorship of the Date class and derived classes. A special thanks to the GPFARM Team, especially Decision Support System programmers, Debbie Edmunds and Bruce Vandenberg, whose collaboration contributed to the framework's successful application in GPFARM 1.0. Thanks to Charlie Bay for the instruction in OOA, OOD and OOP resulting in a successful application.

## References

Ahuja, L.R., Johnsen, K.E., Rojas, K.W., 1999. Water and chemical transport in soil matrix and macropores, ch. 2. In: Ahuja, et al. (Eds.), Root Zone Water Quality Model: Modelling Management Effects on Water Quality and Crop Production, Water Resources Publications, LLC, Highlands Ranch, Colorado, pp. 13–50 (in press).

Arnold, J.G., Weltz, M.A., Alberts, E.E., Flanagan, D.C., 1995. Plant growth component (ch. 8). In: Flanagan, D.C., Nearing, M.A. II (Eds.), USDA-Water Erosion Prediction Project: Hillslope Profile and Watershed Model Technical Documentation. NSERL Report No. 101-8.41. USDA-ARS National Soil Erosion Research Laboratory, West Lafayette, IN, pp. 8.1–8.41.

Ascough, J.C. II, Baffaut, C., Nearing, M.A., Liu, B.Y., 1997. The WEPP watershed model: I. Hydrology and erosion. Trans. ASAE 40 (4), 921–933.

Ascough, J.C. II, Baffaut, C., Nearing, M.A., Flanagan, D.C., 1995. Watershed model channel hydrology and erosion processes (ch. 13). In: Flanagan, D.C., Nearing, M.A. (Eds.), USDA-Water Erosion Prediction Project: Hillslope Profile and Watershed Model Technical Documentation. NSERL Report No. 10. USDA-ARS National Soil Erosion Research Laboratory, West Lafayette, IN, pp. 13.1–13.20.

Ascough II, J.C., Shaffer, M.J., Hoag, D.L., McMaster, G.S., Ahuja, L.R., 2000. GPFARM: An integrated decisions support systems for sustainable Great Plains Agriculture. Proceedings of the 10$^{th}$ International Soil Conservation Organization Conference (ISCO): Sustaining the Global Farm —

Local Action for Land Leadership, Purdue University, West Lafayette, IN, May 23–28, 1999 (in press).

Baker, B.B, Bourdon, R.M., Hanson, J.D., 1992. FORAGE: a simulation model of grazing behavior in beef cattle. Ecol. Model. 60, 257–279.

Beckie, H.J., Moullin, A.P., Campbell, C.A., Brandt, S.A., 1995. Testing effectiveness of four simulation models for estimating nitrates and water in two soils. Can. J. Soil Sci. 75, 135–143.

Buick, R.D., Stone, N.D., Scheckler, R.K., Roach, J.W., 1992. CROPS: a whole-farm crop rotation planning system to implement sustainable agriculture. AI Applic. 6 (3), 29–50.

Booch, G., 1994. The object model (ch. 2). In: Object-Oriented Analysis and Design with Applications, 2nd edn. Benjamin/Cummins Publishing Co., Inc, Redwood City, CA.

Canner, S.R., Wiles, L.J., Dunan, C.R., Erskine, R.H., 1998. A new approach for modeling long term multi-species weed population dynamics. Proc. Western Weed Sci. Soc. 51, 30.

Coad, P., Yourdon, E., 1991. Object-Oriented Analysis. Prentice-Hall, Englewood Cliffs, NJ, p. 232.

Crosby, C.J., 1990. A simulation modeling tool for nitrogen dynamics using object-oriented programming. AI Applic. Nat. Res. Manage. 4 (2), 94–100.

Engel, T., Hoogenboom, G., Jones, J.W., Wilkens, P.W., 1997. AEGIS/WIN: a computer program for the application of crop simulation models across geographic areas. Agron. J. 89 (6), 919–928.

Farahani, H.J., Ahuja, L.R., 1996. Evapotranspiration modeling of partial canopy/residue covered fields. Trans. ASAE 39 (6), 2051–2064.

Farahani, H.J., DeCoursey, D.G., 1999. Potential evaporation and transpiration processes in the soil-residue-canopy system, ch. 3. In: Ahuja, et al. (Eds.), Root Zone Water Quality Model: Modelling Management Effects on Water Quality and Crop Production, Water Resources Publications, LLC, Highlands Ranch, Colorado, pp. 51–80 (in press).

Freeman, S.A., 1992. Object-oriented methodology for analyzing and allocating resources for field operations. Appl. Eng. Agric. 8 (4), 525–535.

Gauthier, L., Neel, T., 1996. SAGE: an object-oriented framework for the construction of farm decision support systems. Comput. Electron. Agric. 16 (1), 1–20.

Hansen, S., Shaffer, M.J., Jensen, H.E., 1995. Developments in modelling nitrogen transformations in soil, ch. 3. In: Nitrogen Fertilization and the Environment. Dekker, New York, pp. 83–107.

Hanson, J.D., Baker, B.B., Bourdon, R.M. 1992. SPUR2 Documentation and User Guide. GPSR Technical Report No. 1. 46 pp.

Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G., 1992. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, Reading, MA.

Khakural, B.R., Robert, P.C., 1993. Soil nitrate leaching potential indices: using a simulation model as a screening system. J. Environ. Qual. 22, 839–845.

Lal, H., 1991. An object-oriented field operations simulator in PROLOG. Trans. ASAE 34 (3), 1031–1039.

Nachabe, M.H., Ahuja, L.R., 1996. Quasi-analytical solution for predicting the redistribution of surface-applied chemicals. Trans. ASAE 39, 1659–1664.

Pannel, D.J., 1996. Lessons from a decade of whole-farm modelling in Western Australia. Rev. Agric. Econ. 18 (3), 373–383.

Power, J.M., 1993. Object-oriented design of decision support systems in natural resource management. Comput. Electron. Agric. 8 (4), 301–324.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., 1991. Object-oriented modeling and design. Prentice-Hall, Englewood Cliffs, NJ.

Schilizzi, G.M., Boulier, F., 1997. Agric. Syst. 54 (4), 477–499.

Sequeira, R.A., Makela, M.E., El-Zik, K.M., Stone, N.D. 1990. Coughing object-oriented plant and Heliothis models. Proceedings—Beltwide cotton Production Research Conferences, 339–342.

Shaffer, M.J., Brodahl, M.K., 1998. Rule-based management for simulation in agricultural decision support systems. Comput. Electron. Agric. 21, 135–152.

Shaffer, M.J, Halvorson, A.D., Pierce, F.J., 1991. Nitrate leaching and economic analysis package (NLEAP): model description and application (ch. 13). In: Follett, R.F., Keeney, D.R., Cruse, R.M. (Eds.), Managing Nitrogen for Groundwater Quality and Farm Profitability. Soil Science Society of America, Madison, WI, pp. 285–322.

Shaffer, M.J., Wylie, B.K., Follett, R.F., Bartling, P.N.S., 1994. Using climate/weather data with the NLEAP model to manage soil nitrogen. Agric. Forest Meteor. 69, 111–123.

Shaffer, M.J., Wylie, B.K., Hall, M.D., 1995. Identification and mitigation of nitrate leaching hot spots using NLEAP-GIS technology. J. Contaminant Hydrol. 20 (1995), 253–263.

Shaw, M., 1984. Abstraction techniques in modern programming languages. IEEE Software 1 (4), 10.

Silvert, W., 1993. Object-oriented ecosystem modelling. Ecol. Model. 68 (1993), 91–118.

Van Evert, F.K., Campbell, G.S., 1994. CropSyst: a collection of object oriented simulation models of agricultural systems. Agron. J. 86 (2), 325–331.

Williams, J.R., Jones, C.A., Dyke, P.T., 1984. A modeling approach to determining the relationship between erosion and soil productivity. Trans. ASAE 27, 129–144.

Wiles, L.J., King, R.P., Schweizer, E.E., Lybecker, D.W., Swinton, S.M., 1996. GWM: Gen. Weed Manage. Model. 50, 355–376.